PostgreSQL

QUERY PERFORMANCE

Ioan Marar (Johnny)
Senior Data Engineer – Betfair Romania
j_marar@yahoo.com

PostgreSQL - AGENDA

- Overview of Database
  - What is PostgreSQL?
  - Aurora overview
- Best Practices
  - Indexes
  - Statistics
  - Explain Plan
  - Partitions
- Troubleshooting
- Q & A

What is PostgreSQL?

- The PostgreSQL is a relational database.
- It is fully ACID (Atomicity, Consistency, Isolation, Durability) complaint and MVCC (Multi-Version Concurrency Control).
- Postgres offers the read committed and serializable isolation levels.
- The World’s Most Advanced Open Source Relational Database
- More details:

  https://www.postgresql.org/

Aurora overview

- Amazon Aurora is a relational database service (RDS) on AWS that combines the speed and availability of high-end commercial databases

- Aurora delivers up to 3X the throughput of standard PostgreSQL running on the same hardware, enabling existing PostgreSQL applications and tools to run without requiring modification.

- More details:

  https://aws.amazon.com/rds/aurora/details/postgresql-details/

Best Practices – **Indexes**

- **Why is my query not using an index?**
  - Most of the time, the planner chooses correctly.
  - Kept the statistics up to date.

```
VACUUM ANALYZE bet;
```

  - The index is not used when a function is applying over the index column: convert, like (%pattern%), date function…
  - Sample:

- where: timezone('UTC', placed_date_time) >= (localtimestamp - interval '1 days') - not use the index

- some_string LIKE 'pattern%' - use the index

- some_string LIKE '%pattern%' - not use the index

Best Practices – **Statistics**

- **What are statistics?**
  - pg_stat_database
  - pg_stat_user_tables
  - pg_stat_user_indexes

```
select * from pg_stat_user_tables where schemaname = 'ods_exchange' and relname like 'tbl_bets_settled_20200215%';
```

relname	seq_scan	seq_tup_read	idx_scan	idx_tup_fetch	n_tup_ins	n_tup_upd	n_tup_del	n_live_tup	n_dead_tup
tbl_bets_settled_20200215	815	20277436127	631405242	5776792613	100488445	7485	0	100470318	6430

last_vacuum	last_autovacuum	last_analyze	last_autoanalyze	vacuum_count	autovacuum_count	analyze_count	autoanalyze_count
(null)	(null)	(null)	2020-02-15 22:21:17.783	0	0	0	61

Best Practices – **Statistics**

- pg_stat_user_indexes

```
select * from pg_stat_user_indexes where relname = 'tbl_bets_settled_20200215';
```

relname	indexrelname	idx_scan	idx_tup_read	idx_tup_fetch
tbl_bets_settled_20200215	tbl_bets_settled_20200215_settled_date	79426	5843512133	5836898717
tbl_bets_settled_20200215	tbl_bets_settled_20200215_account_id	2074039	918639	917006
tbl_bets_settled_20200215	idx_ods_exchange_tbl_bets_settled_20200215_bet_id	473667	55019	54802
tbl_bets_settled_20200215	pk_tbl_bets_settled_20200215	628785692	390144696	385962531

Best Practices – **Explain Plan**

- The EXPLAIN command is used to see what query plan the planner creates for any query. It is based on estimated execution.

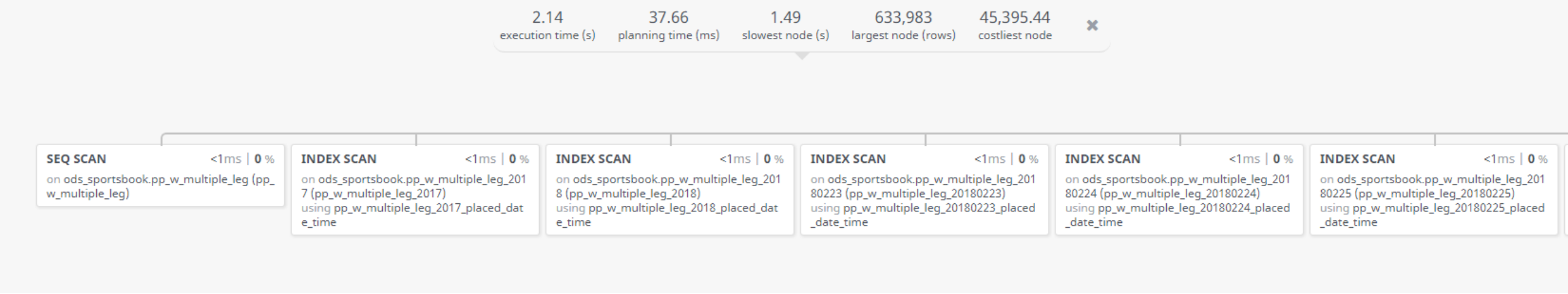
- Estimated start-up cost.
- Estimated total cost.
- Estimated number of rows output by this plan node.
- Estimated average width of rows output by this plan node (in bytes).

- The JSON could be analyzed on the below web page:

http://tatiyants.com/pev/#/plans/new

Best Practices – **Explain Plan**

Sample:
EXPLAIN (ANALYZE, COSTS, VERBOSE, BUFFERS, FORMAT JSON)
select * from leg where placed_date_time >= (localtimestamp - interval '1 days')



http://tatiyants.com/pev/#/plans/new

always prefer doing an INNER JOIN instead of a LEFT OUTER JOIN, when this is possible

Best Practices – **Explain Plan**

- Sample:

```
EXPLAIN SELECT * FROM bet WHERE placed_date_time >= (localtimestamp - interval '1 days')
```

- Append (cost=0.00..48954.99 rows=635240 width=364)

...

- -> Index Scan using bet_20200123_placed_date_time on bet_20200123 (cost=0.43..8.36 rows=1 width=364)
  Index Cond: (placed_date_time >= (('now'::cstring)::timestamp without time zone - '1 day'::interval))

always prefer doing an INNER JOIN instead of a LEFT OUTER JOIN, when this is possible

Best Practices – **Explain Plan**

```
EXPLAIN SELECT * FROM bet WHERE timezone('UTC', placed_date_time) >= (localtimestamp - interval '1 days')
```

- Append (cost=0.00..7078256.45 rows=31328171 width=376)
- .....
- -> Seq Scan on bet_20200123 (cost=0.00..49174.07 rows=217231 width=364)
  Filter: (timezone('UTC'::text, placed_date_time) >= (('now'::cstring)::timestamp without time zone - '1 day'::interval))

Best Practices – **Indexes**

- **Indexes on temporary tables**
  - It is recommended to create the indexes on the temporary tables in the query when these tables are used in join with other tables
  - Usually the indexes are created on the columns using in join condition
  - After the indexes were created it is recommended to ANALYZE the temporary tables
-

Best Practices – **Indexes on temp table**

```
CREATE TEMP TABLE tmp_bet as
SELECT mult_id, placed_date_time, settled_date_time
FROM bet WHERE placed_date_time >= (localtimestamp - interval '1 days')
```

```
EXPLAIN
  SELECT * FROM tmp_bet b
  JOIN leg l ON b.mult_id = l.mult_id
  WHERE b.mult_id = '869414731'
```

```
->  Seq Scan on tmp_bet b  (cost=0.00..9780.21 rows=1 width=28)
      Filter: ((mult_id)::text = '869414731'::text)
->  Append  (cost=0.00..5578.50 rows=1292 width=422)
      ->  Seq Scan on leg_1  (cost=0.00..0.00 rows=1 width=2800)
            Filter: ((mult_id)::text = '869414731'::text)
      ->  Index Scan using leg_mult_id_leg_2019 on leg_2019_1_1  (cost=0.14..8.15 rows=1 width=2800)
            Index Cond: ((mult_id)::text = '869414731'::text)
      ->  Index Scan using leg_mult_id_leg_20190813 on leg_20190813_1_2  (cost=0.42..35.71 rows=8 width=436)
```

Best Practices – **Indexes on temp table**

```
CREATE INDEX IX_tmp_bet_mult_id on tmp_bet(mult_id);
ANALYZE tmp_bet;
```

```
->  Index Scan using ix tmp bet mult id on tmp bet b  (cost=0.42..8.44 rows=1 width=28)
      Index Cond: ((mult_id)::text = '869414731'::text)
->  Append  (cost=0.00..5578.50 rows=1292 width=422)
      ->  Seq Scan on leg 1  (cost=0.00..0.00 rows=1 width=2800)
            Filter: ((mult_id)::text = '869414731'::text)
      ->  Index Scan using leg_mult_id_leg_2019 on leg_2019 1_1  (cost=0.14..8.15 rows=1 width=2800)
            Index Cond: ((mult_id)::text = '869414731'::text)
```

Best Practices – **Partition Tables**

- Do not use the PostgreSql default partition because it is based on trigger insert, which makes it's performance poor.
- Large tables are partitioned to improve query performance and assist in data archiving.
- Postgres partitioning works by creating a "master" table, which holds no data, and numerous "child" partition tables which inherit their schema from the master.
- **It is very important the query uses the partition column in the where/join condition.**

Best Practices – **Partition Tables**

- **Partition Tables on PostgreSQL**

```
CREATE TABLE
bet_20200123
(
    CONSTRAINT pk_bet_2020123 PRIMARY KEY (mult_id,
    source_product_platform_id, placed_date_time),
    CONSTRAINT bet_20200123_placed_date_time_check CHECK ((placed_date_time >=
    '2020-01-23 00:00:00'::TIMESTAMP without TIME zone)
AND (
        placed_date_time < '2020-01-24 00:00:00'::TIMESTAMP without TIME zone))
)
INHERITS
(
    bet
);
```

Best Practices – **Join tables**

- joins are also a better solution than subqueries
- start the query from a small table and continue with large table in JOIN
- use in join condition the indexes column and partition column
- do not use select * , try to select only the useful columns
- always prefer doing an INNER JOIN instead of a LEFT OUTER JOIN, when this is possible
- use EXPLAIN to analyze the query cost, indexes used

Best Practices – **Troubleshooting**

- Use the partition column and indexes in the join condition and where condition.
- CTEs and sub queries can be slower than using a series of temp tables.
- Watch out for SEQ SCAN where INDEX SCAN should be used.
- Create indexes on temp tables if required.

Best Practices – **Troubleshooting**

- Do not use the conversion data type on the columns use in join and where conditions. The indexes are not use in this case.

- Use the same date filter on each historical table.

- *Don't be afraid to try different approaches to see what is best for your particular requirement. If you don't know if joining tables will be faster than populating a temp table first, try both and see which works out best.*

Q&A